LEARNING TO COMPOSE WORDS INTO SENTENCES WITH REINFORCEMENT LEARNING

Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, Wang Ling
DeepMind
{dyogatama,pblunsom,cdyer,etg,lingwang}@google.com

ABSTRACT

We use reinforcement learning to learn tree-structured neural networks for computing representations of natural language sentences. In contrast with prior work on tree-structured models in which the trees are either provided as input or predicted using supervision from explicit treebank annotations, the tree structures in this work are optimized to improve performance on a downstream task. Experiments demonstrate the benefit of learning task-specific composition orders, outperforming both sequential encoders and recursive encoders based on treebank annotations. We analyze the induced trees and show that while they discover some linguistically intuitive structures (e.g., noun phrases, simple verb phrases), they are different than conventional English syntactic structures.

1 INTRODUCTION

Languages encode meaning in terms of hierarchical, nested structures on sequences of words (Chomsky, 1957). However, the degree to which neural network architectures that compute representations of the meaning of sentences for practical applications should explicitly reflect such structures is a matter for debate.

There are three predominant approaches for constructing vector representations of sentences from a sequence of words. The first composes words sequentially using a recurrent neural network, treating the RNN's final hidden state as the representation of the sentence (Cho et al., 2014; Sutskever et al., 2014; Kiros et al., 2015). In such models, there is no explicit hierarchical organization imposed on the words, and the RNN's dynamics must learn to simulate it. The second approach uses tree-structured networks to recursively compose representations of words and phrases to form representations of larger phrases and, finally, the complete sentence. In contrast to sequential models, these models' architectures are organized according to each sentence's syntactic structure, that is, the hierarchical organization of words into nested phrases that characterizes human intuitions about how words combine to form grammatical sentences. Prior work on tree-structured models has assumed that trees are either provided together with the input sentences (Clark et al., 2008; Grefenstette & Sadrzadeh, 2011; Socher et al., 2011; 2013; Tai et al., 2015) or that they are predicted based on explicit treebank annotations jointly with the downstream task (Bowman et al., 2016; Dyer et al., 2016). The last approach for constructing sentence representations uses convolutional neural networks to produce the representation in a bottom up manner, either with syntactic information (Ma et al., 2015) or without (Kim, 2014; Kalchbrenner et al., 2014).

Our work can be understood as a compromise between the first two approaches. Rather than using explicit supervision of tree structure, we use reinforcement learning to learn tree structures (and thus, sentence-specific compositional architectures), taking performance on a downstream task that uses the computed sentence representation as the reward signal. In contrast to sequential RNNs, which ignore tree structure, our model still generates a latent tree for each sentence and uses it to structure the composition. Our hypothesis is that encouraging the model to learn tree-structured compositions will bias the model toward better generalizations about how words compose to form sentence meanings, leading to better performance on downstream tasks.

oh, check this out

This work is related to unsupervised grammar induction (Klein & Manning, 2004; Blunsom & Cohn, 2010; Spitkovsky et al., 2011, *inter alia*), which seeks to infer a generative grammar of an infinite language from a finite sample of strings from the language—but without any semantic feedback.

Previous work on unsupervised grammar induction that incorporates semantic supervision involves designing complex models for Combinatory Categorial Grammars (Zettlemoyer & Collins, 2005). Since semantic feedback has been proposed as crucial for the acquisition of syntax (Pinker, 1984), our model offers a simpler alternative.[1] However, our primary focus is on improving performance on the downstream model, so the learner may settle on a different solution than conventional English syntax. We thus also explore what kind of syntactic structures are derivable from shallow semantics.

Experiments on various tasks (i.e., sentiment analysis, semantic relatedness, natural language inference, and sentence generation) show that reinforcement learning is a promising direction to discover hierarchical structures of sentences. Notably, representations learned this way outperformed both conventional left-to-right models and tree-structured models based on linguistic syntax in downstream applications. This is in line with prior work showing the value of learning tree structures in statistical machine translation models (Chiang, 2007). Although the induced tree structures manifested a number of linguistically intuitive structures (e.g., noun phrases, simple verb phrases), there are a number of marked differences to conventional analyses of English sentences (e.g., an overall left-branching structure).

2 Model

Our model consists of two components: a sentence representation model and a reinforcement learning algorithm to learn the tree structure that is used by the sentence representation model.

2.1 Tree LSTM

Our sentence representation model follows the Stack-augmented Parser-Interpreter Neural Network (SPINN; Bowman et al., 2016), SPINN is a shift-reduce parser that uses Long Short-Term Memory (LSTM; Hochreiter and Schmidhuber, 1997) as its composition function. Given an input sentence of N words $\mathbf{x} = \{x_1, x_2, \dots, x_N\}$, we represent each word by its embedding vector $\mathbf{x}_i \in \mathbb{R}^D$. The parser maintains an index pointer p starting from the leftmost word ($p = 1$) and a stack. To parse the sentence, it performs a sequence of operations $\mathbf{a} = \{a_1, a_2, \dots, a_{2N-1}\}$, where $a_t \in \{\text{SHIFT}, \text{REDUCE}\}$. A SHIFT operation pushes $\mathbf{x}_p$ to the stack and moves the pointer to the next word (p_{++}); while a REDUCE operation pops two elements from the stack, composes them to a single element, and pushes it back to the stack. SPINN uses Tree LSTM (Tai et al., 2015) as the REDUCE composition function, which we follow. In Tree LSTM, each element of the stack is represented by two vectors, a hidden state representation $\mathbf{h}$ and a memory representation $\mathbf{c}$. Two elements of the stack $(\mathbf{h}_i, \mathbf{c}_i)$ and $(\mathbf{h}_j, \mathbf{c}_j)$ are composed as:

Tree-LSTM: This is it

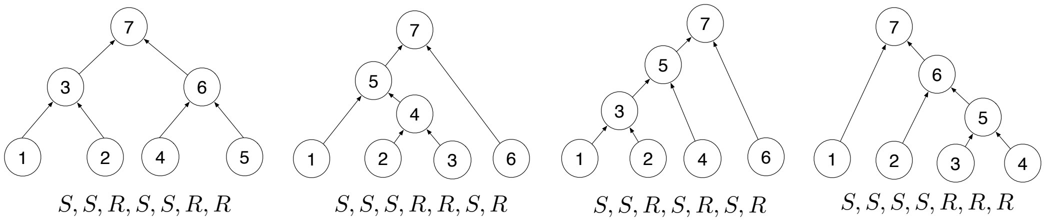
$$\begin{aligned}
\mathbf{i} &= \sigma(\mathbf{W}_I[\mathbf{h}_i, \mathbf{h}_j] + \mathbf{b}_I) \\
\mathbf{f}_L &= \sigma(\mathbf{W}_{F_L}[\mathbf{h}_i, \mathbf{h}_j] + \mathbf{b}_{F_L}) \\
\mathbf{f}_R &= \sigma(\mathbf{W}_{F_R}[\mathbf{h}_i, \mathbf{h}_j] + \mathbf{b}_{F_R}) \\
\mathbf{o} &= \sigma(\mathbf{W}_O[\mathbf{h}_i, \mathbf{h}_j] + \mathbf{b}_I) \\
\mathbf{g} &= \tanh(\mathbf{W}_G[\mathbf{h}_i, \mathbf{h}_j] + \mathbf{b}_G) \\
\mathbf{c} &= \mathbf{f}_L \odot \mathbf{c}_i + \mathbf{f}_R \odot \mathbf{c}_j + \mathbf{i} \odot \mathbf{g} \\
\mathbf{h} &= \mathbf{o} \odot \mathbf{c}
\end{aligned} \tag{1}$$

where $[\mathbf{h}_i, \mathbf{h}_j]$ denotes concatenation of $\mathbf{h}_i$ and $\mathbf{h}_j$, and σ is the sigmoid activation function.

A unique sequence of $\{\text{SHIFT}, \text{REDUCE}\}$ operations corresponds to a unique binary parse tree of the sentence. A SHIFT operation introduces a new leaf node in the parse tree, while a REDUCE operation combines two nodes by merging them into a constituent. See Figure 1 for an example. We note that for a sentence of length N, there are exactly N SHIFT operations and $N-1$ REDUCE operations that are needed to produce a binary parse tree of the sentence. The final sentence representation produced by the Tree LSTM is the hidden state of the final element of the stack $\mathbf{h}_{N-1}$ (i.e., the topmost node of the tree).

Tracking LSTM SPINN optionally augments Tree LSTM with another LSTM that incorporates contextual information in sequential order called tracking LSTM, which has been shown to improve

[1] Our model only produces an interpretation grammar that parses language instead of a generative grammar.



S, S, R, S, S, R, R $\quad$ S, S, S, R, R, S, R $\quad$ S, S, R, S, R, S, R $\quad$ S, S, S, S, R, R, R

Figure 1: Four examples of trees and their corresponding SHIFT (S) and REDUCE (R) sequences. In each of the examples, there are 4 input words (4 leaf nodes), so 7 operations (4 S, 3 R) are needed to construct a valid tree. The nodes are labeled with the timesteps in which they are introduced to the trees $t \in \{1, \dots, 7\}$. A SHIFT operation introduces a leaf node, whereas a REDUCE operation introduces a non-leaf node by combining two previously introduced nodes. We can see that different S-R sequences lead to different tree structures.

performance for textual entailment. It is a standard recurrent LSTM network that takes as input the hidden states of the top two elements of the stack and the embedding vector of the word indexed by the pointer at timestep t. Every time a REDUCE operation is performed, the output of the tracking LSTM $\mathbf{e}$ is included as an additional input in Eq. 1 (i.e., the input to the REDUCE composition function is $[\mathbf{h}_i, \mathbf{h}_j, \mathbf{e}]$ instead of $[\mathbf{h}_i, \mathbf{h}_j]$).

2.2 Reinforcement Learning

In previous work (Tai et al., 2015; Bowman et al., 2016), the tree structures that guided composition orders of Tree LSTM models are given directly as input (i.e., $\mathbf{a}$ is observed and provided as an input). Formally, each training data is a triplet $\{\mathbf{x}, \mathbf{a}, \mathbf{y}\}$. Tai et al. (2015) consider models where $\mathbf{a}$ is also given at test time, whereas Bowman et al. (2016) explore models where $\mathbf{a}$ can be either observed or not at test time. When it is only observed during training, a policy is trained to predict $\mathbf{a}$ at test time. Note that in this case the policy is trained to match explicit human annotations (i.e., Penn TreeBank annotations), so the model learns to optimize representations according to structures that follows human intuitions. They found that models that observe $\mathbf{a}$ at both training and test time are better than models that only observe $\mathbf{a}$ during training.

Our main idea is to use reinforcement learning (policy gradient methods) to discover the best tree structures for the task that we are interested in. We do not place any kind of restrictions when learning these structures other than that they have to be valid binary parse trees, so it may result in tree structures that match human linguistic intuition, heavily right or left branching, or other solutions if they improve performance on the downstream task.

We parameterize each action $a \in \{\text{SHIFT}, \text{REDUCE}\}$ by a policy network $\pi(a \mid \mathbf{s}; \mathbf{W}_R)$, where $\mathbf{s}$ is a representation of the current state and $\mathbf{W}_R$ is the parameter of the network. Specifically, we use a two-layer feedforward network that takes the hidden states of the top two elements of the stack $\mathbf{h}_i$ and $\mathbf{h}_j$ and the embedding vector of the word indexed by the pointer $\mathbf{x}_p$ as its input:

$$\mathbf{s} = \text{ReLU}(\mathbf{W}_R^1[\mathbf{h_i}, \mathbf{h_j}, \mathbf{x_p}] + \mathbf{b}_R^1)$$
$$\pi(a \mid \mathbf{s}; \mathbf{W}_R) \propto \exp(\mathbf{w}_R^{2\top} \mathbf{s} + b_R^2)$$

where $[\mathbf{h}_i, \mathbf{h}_j, \mathbf{x}_p]$ denotes concatenation of vectors inside the brackets.

If $\mathbf{a}$ is given as part of the training data, the policy network can be trained—in a supervised training regime—to predict actions that result in trees that match human intuitions. Our training data, on the other hand, is a tuple $\{\mathbf{x}, \mathbf{y}\}$. We use REINFORCE (Williams, 1992), which is an instance of a broader class of algorithms called policy gradient methods, to learn $\mathbf{W}_R$ such that the sequence of actions $\mathbf{a} = \{a_1, \dots, a_T\}$ maximizes:

$$\mathcal{R}(\mathbf{W}) = \mathbb{E}_{\pi(\mathbf{a}, \mathbf{s}; \mathbf{W}_R)} \left[\sum_{t=1}^{T} r_t a_t \right],$$

where r_t is the reward at timestep t. We use performance on a downstream task as the reward function. For example, if we are interested in using the learned sentence representations in a classification

Table 1: Descriptive statistics of datasets used in our experiments.

Dataset	# of train	# of dev	# of test	Vocab size
SICK	4,500	500	4,927	2,172
SNLI	550,152	10,000	10,000	18,461
SST	98,794	872	1,821	8,201
IMDB	441,617	223,235	223,236	29,209

task, our reward function is the probability of predicting the correct label using a sentence representation composed in the order given by the sequence of actions sampled from the policy network, so $\mathcal{R}(\mathbf{W}) = \log p(y \mid \text{T-LSTM}(\mathbf{x}); \mathbf{W})$, where we use $\mathbf{W}$ to denote all model parameters (Tree LSTM, policy network, and classifier parameters), y is the correct label for input sentence $\mathbf{x}$, and $\mathbf{x}$ is represented by the Tree LSTM structure in §2.1. For a natural language generation task where the goal is to predict the next sentence given the current sentence, we can use the probability of predicting words in the next sentence as the reward function, so $\mathcal{R}(\mathbf{W}) = \log p(\mathbf{x}_{s+1} \mid \text{T-LSTM}(\mathbf{x}_s); \mathbf{W})$.

Note that in our setup we do not immediately receive a reward after performing an action at timestep t. The reward is only observed at the end after we finish creating a representation for the current sentence with Tree LSTM and use the resulting representation for the downstream task. At each timestep t, we sample a valid action according to $\pi(a \mid \mathbf{s}; \mathbf{W}_R)$. We add two simple constraints to make the sequence of actions result in a valid tree: REDUCE is forbidden if there are fewer than two elements on the stack, and SHIFT is forbidden if there are no more words to read from the sentence. After reaching timestep $2N - 1$, we construct the final representation and receive a reward that is used to update our model parameters.

We experiment with two learning methods: unsupervised structures and semi-supervised structures. Suppose that we are interested in a classification task. In the unsupervised case, the objective function that we maximize is $\log p(y \mid \text{T-LSTM}(\mathbf{x}); \mathbf{W})$. In the semi-supervised case, the objective function for the first E epochs also includes a reward term for predicting the correct SHIFT or REDUCE actions obtained from an external parser—in addition to performance on the downstream task, so we maximize $\log p(y \mid \text{T-LSTM}(\mathbf{x}); \mathbf{W}) + \log \pi(\mathbf{a} \mid \mathbf{s}; \mathbf{W}_R)$. The motivation behind this model is to first guide the model to discover tree structures that match human intuitions, before letting it explore other structures close to these ones. After epoch E, we remove the second term from our objective function and continue maximizing the first term. Note that unsupervised and semi-supervised here refer to the tree structures, not the nature of the downstream task.

3 Experiments

3.1 Baselines

The goal of our experiments is to evaluate our hypothesis that we can discover useful task-specific tree structures (composition orders) with reinforcement learning. We compare the following composition methods (the last two are unique to our work):

- **Right to left**: words are composed from right to left.[2]
- **Left to right**: words are composed from left to right. This is the standard recurrent neural network composition order.
- **Bidirectional**: A bidirectional right to left and left to right models, where the final sentence embedding is an average of sentence embeddings produced by each of these models.
- **Supervised syntax**: words are composed according to a predefined parse tree of the sentence. When parse tree information is not included in the dataset, we use Stanford parser (Klein & Manning, 2003) to parse the corpus.
- **Semi-supervised syntax**: a variant of our reinforcement learning method, where for the first E epochs we include rewards for predicting predefined parse trees given in the super-

[2] We choose to include right to left as a baseline since a right-branching tree structure—which is the output of a right to left composition order—has been shown to be a reliable baseline for unsupervised grammar induction. (Klein & Manning, 2004)

vised model, before letting the model explore other kind of tree structures at later epochs (i.e., semi-supervised structures in §2.2).

- **Latent syntax**: another variant of our reinforcement learning method where there is no predefined structures given to the model at all (i.e., unsupervised structures in §2.2).

For learning, we use stochastic gradient descent with minibatches of size 1 and ℓ_2 regularization constant tune on development data from $\{10^{-4}, 10^{-5}, 10^{-6}, 0\}$. We use performance on development data to choose the best model and decide when to stop training.

3.2 Tasks

We evaluate our method on four sentence representation tasks: sentiment classification, semantic relatedness, natural language inference (entailment), and sentence generation. We show statistics of the datasets in Table 1 and describe each task in details in the followings.

Stanford Sentiment Treebank We evaluate our model on a sentiment classification task from the Stanford Sentiment Treebank (Socher et al., 2013). We use the binary classification task where the goal is to predict whether a sentence is a positive or a negative movie review.

We set the word embedding size to 100 and initialize them with Glove vectors (Pennington et al., 2014)[3]. For each sentence, we create a 100-dimensional sentence representation $\mathbf{s} \in \mathbb{R}^{100}$ with Tree LSTM, project it to a 200-dimensional vector and apply ReLU: $\mathbf{q} = \text{ReLU}(\mathbf{W}_p\mathbf{s} + \mathbf{b}_p)$, and compute $p(\hat{y} = c \mid \mathbf{q}; \mathbf{w}_q) \propto \exp(\mathbf{w}_{q,c}\mathbf{q} + b_q)$.

We run each model 3 times (corresponding to three different initialization points) and use the development data to pick the best model. We show the results in Table 2. Our results agree with prior work that have shown the benefits of using syntactic parse tree information on this dataset (i.e., supervised recursive model is generally better than sequential models). The best model is the latent syntax model, which is also competitive with results from other work on this dataset. Both the latent and semi-supervised syntax models outperform models with predefined structures, demonstrating the benefit of learning task-specific composition orders.

Table 2: Classification accuracy on Stanford Sentiment Treebank dataset. The number of parameters includes word embedding parameters and is our approximation when not reported in previous work.

Model	Acc.	# params.
100D-Right to left	83.9	1.2m
100D-Left to right	84.7	1.2m
100D-Bidirectional	84.7	1.5m
100D-Supervised syntax	85.3	1.2m
100D-Semi-supervised syntax	86.1	1.2m
100D-Latent syntax	**86.5**	1.2m
RNTN (Socher et al., 2013)	85.4	-
DCNN (Kalchbrenner et al., 2014)	86.8	-
CNN-random(Kim, 2014)	82.7	-
CNN-word2vec (Kim, 2014)	87.2	-
CNN-multichannel (Kim, 2014)	88.1	-
NSE (Munkhdalai & Yu, 2016a)	89.7	5.4m
NTI-SLSTM (Munkhdalai & Yu, 2016b)	87.8	4.4m
NTI-SLSTM-LSTM (Munkhdalai & Yu, 2016b)	89.3	4.8m
Left to Right LSTM (Tai et al., 2015)	84.9	2.8m
Bidirectional LSTM (Tai et al., 2015)	87.5	2.8m
Constituency Tree–LSTM–random (Tai et al., 2015)	82.0	2.8m
Constituency Tree–LSTM–GloVe (Tai et al., 2015)	88.0	2.8m
Dependency Tree-LSTM (Tai et al., 2015)	85.7	2.8m

Semantic relatedness The second task is to predict the degree of relatedness of two sentences from the Sentences Involving Compositional Knowledge corpus (SICK; Marelli et al., 2014) . In this dataset, each pair of sentences are given a relatedness score on a 5-point rating scale. For each sentence, we use Tree LSTM to create its representations. Denote the final representations

[3] `http://nlp.stanford.edu/projects/glove/`

by $\{\mathbf{s}_1, \mathbf{s}_2\} \in \mathbb{R}^{100}$. We construct our prediction by computing: $\mathbf{u} = (\mathbf{s}_2 - \mathbf{s}_1)^2$, $\mathbf{v} = \mathbf{s}_1 \odot \mathbf{s}_2$, $\mathbf{q} = \text{ReLU}(\mathbf{W}_p[\mathbf{u}, \mathbf{v}] + \mathbf{b}_p)$, and $\hat{y} = \mathbf{w}_q^\top \mathbf{q} + b_q$, where $\mathbf{W}_p \in \mathbb{R}^{200\times200}, \mathbf{b}_p \in \mathbb{R}^{200}, \mathbf{w}_q \in \mathbb{R}^{200}, b_q \in \mathbb{R}^1$ are model parameters, and $[\mathbf{u}, \mathbf{v}]$ denotes concatenation of vectors inside the brackets. We learn the model to minimize mean squared error.

We run each model 5 times and use the development data to pick the best model. Our results are shown in Table 3. Similar to the previous task, they clearly demonstrate that learning the tree structures yield to better performance.

We also provide results from other work on this dataset for comparisons. Some of these models (Lai & Hockenmaier, 2014; Jimenez et al., 2014; Bjerva et al., 2014) rely on feature engineering and are designed specifically for this task. Our Tree LSTM implementation performs competitively with most models in terms of mean squared error. Our best model—semi-supervised syntax—is better than most models except LSTM models of Tai et al. (2015) which were trained with a different objective function.[4] Nonetheless, we observe the same trends with their results that show the benefit of using syntactic information on this dataset.

Table 3: Mean squared error on SICK dataset.

Model	MSE	# params.
100D-Right to left	0.461	1.0m
100D-Left to right	0.394	1.0m
100D-Bidirectional	0.373	1.3m
100D-Supervised syntax	0.381	1.0m
100D-Semi-supervised syntax	**0.320**	1.0m
100D-Latent syntax	0.359	1.0m
Illinois-LH (Lai & Hockenmaier, 2014)	0.369	-
UNAL-NLP(Jimenez et al., 2014)	0.356	-
Meaning Factory (Bjerva et al., 2014)	0.322	-
DT-RNN (Socher et al., 2014)	0.382	-
Mean Vectors (Tai et al., 2015)	0.456	650k
Left to Right LSTM (Tai et al., 2015)	0.283	1.0m
Bidirectional LSTM (Tai et al., 2015)	0.274	1.0m
Constituency Tree-LSTM (Tai et al., 2015)	0.273	1.0m
Dependency Tree-LSTM (Tai et al., 2015)	0.253	1.0m

Stanford Natural Language Inference We next evaluate our model for natural language inference (i.e., recognizing textual entailment) using the Stanford Natural Language Inference corpus (SNLI; Bowman et al., 2015) . Natural language inference aims to predict whether two sentences are *entailment*, *contradiction*, or *neutral*, which can be formulated as a three-way classiciation problem. Given a pair of sentences, similar to the previous task, we use Tree LSTM to create sentence representations $\{\mathbf{s}_1, \mathbf{s}_2\} \in \mathbb{R}^{100}$ for each of the sentences. Following Bowman et al. (2016), we construct our prediction by computing: $\mathbf{u} = (\mathbf{s}_2 - \mathbf{s}_1)^2$, $\mathbf{v} = \mathbf{s}_1 \odot \mathbf{s}_2$, $\mathbf{q} = \text{ReLU}(\mathbf{W}_p[\mathbf{u}, \mathbf{v}, \mathbf{s}_1, \mathbf{s}_2] + \mathbf{b}_p)$, and $p(\hat{y} = c \mid \mathbf{q}; \mathbf{w}_q) \propto \exp(\mathbf{w}_{q,c}\mathbf{q} + b_q)$, where $\mathbf{W}_p \in \mathbb{R}^{200\times400}, \mathbf{b}_p \in \mathbb{R}^{200}, \mathbf{w}_q \in \mathbb{R}^{200}, b_q \in \mathbb{R}^1$ are model parameters. The objective function that we maximize is the log likelihood of the correct label under the models.

We show the results in Table 4. The latent syntax method performs the best. Interestingly, the sequential left to right model is better than the supervised recursive model in our experiments, which contradicts results from Bowman et al. (2016) that show 300D-LSTM is worse than 300D-SPINN. A possible explanation is that our left to right model has identical number of parameters with the supervised model due to the inclusion of the tracking LSTM even in the left to right model (the only difference is in the composition order), whereas the models in Bowman et al. (2016) have different number of parameters. Due to the poor performance of the supervised model, semi-supervised training does not help on this dataset, although it does significantly close the gap. Our models underperform state-of-the-art models on this dataset that have almost four times the number of parameters. We only experiment with smaller models since tree-based models with dynamic structures (e.g., our semi-supervised and latent syntax models) take longer to train. See §4 for details and discussions about training time.

[4] Our experiments with the regularized KL-divergence objective function (Tai et al., 2015) do not result in significant improvements, so we choose to report results with the simpler mean squared error objective function.

Table 4: Classification accuracy on SNLI dataset.

Model	Acc.	# params.
100D-Right to left	79.1	2.3m
100D-Left to right	80.2	2.3m
100D-Bidirectional	80.2	2.6m
100D-Supervised syntax	78.5	2.3m
100D-Semi-supervised syntax	80.2	2.3m
100D-Latent syntax	**80.5**	2.3m
100D-LSTM (Bowman et al., 2015)	77.6	5.7m
300D-LSTM (Bowman et al., 2016)	80.6	8.5m
300D-SPINN (Bowman et al., 2016)	83.2	9.2m
1024D-GRU (Vendrov et al., 2016)	81.4	15.0m
300D-CNN (Mou et al., 2016)	82.1	9m
300D-NTI (Munkhdalai & Yu, 2016b)	83.4	9.5m
300D-NSE (Munkhdalai & Yu, 2016a)	84.6	8.5m

Sentence generation The last task that we consider is natural language generation. Given a sentence, the goal is to maximize the probability of generating words in the following sentence. This is a similar setup to the Skip Thought objective (Kiros et al., 2015), except that we do not generate the previous sentence as well. Given a sentence, we encode it with Tree LSTM to obtain $\mathbf{s} \in \mathbb{R}^{100}$. We use a bag-of-words model as our decoder, so $p(w_i \mid \mathbf{s}; \mathbf{V}) \propto \exp(\mathbf{v}_i^\top \mathbf{s})$, where $\mathbf{V} \in \mathbb{R}^{100\times 29,209}$ and $\mathbf{v}_i \in \mathbb{R}^{100}$ is the i-th column of $\mathbf{V}$. Using a bag-of-words decoder as opposed to a recurrent neural network decoder increases the importance of producing a better representation of the current sentence, since the model cannot rely on a sophisticated decoder with a language model component to predict better. This also greatly speeds up our training time.

We use IMDB movie review corpus (Diao et al., 2014) for this experiment, The corpus consists of 280,593, 33,793, and 34,029 reviews in training, development, and test sets respectively. We construct our data using the development and test sets of this corpus. For training, we process 33,793 reviews from the original development set to get 441,617 pairs of sentences. For testing, we use 34,029 reviews in the test set (446,471 pairs of sentences). Half of these pairs is used as our development set to tune hyperparamaters, and the remaining half is used as our final test set. Our results in Table 5 further demonstrate that methods that learn tree structures perform better than methods that have fixed structures.

Table 5: Word perplexity on the sentence generation task. We also show perplexity of the model that does not condition on the previous sentence (unconditional) when generating bags of words for comparison.

Model	Perplexity	# params.
100D-Unconditional	105.6	30k
100D-Right to left	101.4	6m
100D-Left to right	101.1	6m
100D-Bidirectional	100.2	6.2m
100D-Supervised syntax	100.8	6m
100D-Semi-supervised syntax	**98.4**	6m
100D-Latent syntax	99.0	6m

4 Discussion

Learned Structures Our results in §3 show that our proposed method outperforms competing methods with predefined composition order on all tasks. The right to left model tends to perform worse than the left to right model. This suggests that the left to right composition order, similar to how human reads in practice, is better for neural network models. Our latent syntax method is able to discover tree structures that work reasonably well on all tasks, regardless of whether the task is better suited for a left to right or supervised syntax composition order.

We inspect what kind of structures the latent syntax model learned and how closely they match human intuitions. We first compute unlabeled bracketing F_1 scores[5] for the learned structures and

[5] We use evalb toolkit from `http://nlp.cs.nyu.edu/evalb/`.

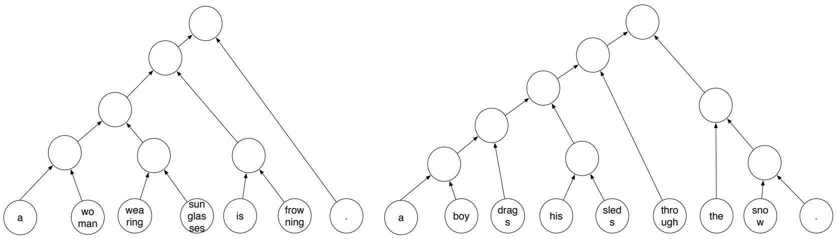


Figure 2: Examples of tree structures learned by our model which show that the model discovers simple concepts such as noun phrases and verb phrases.

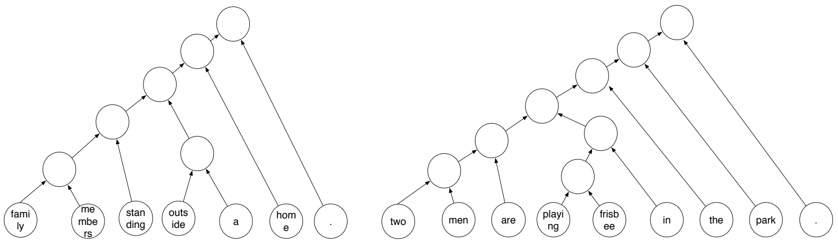


Figure 3: Examples of unconventional tree structures.

parses given by Stanford parser on SNLI and Stanford Sentiment Treebank. In the SNLI dataset, there are 10,000 pairs of test sentences (20,000 sentences in total), while the Stanford Sentiment Treebank test set contains 1,821 test sentences. The F_1 scores for the two datasets are 41.73 and 40.51 respectively. For comparisons, F_1 scores of a right (left) branching tree are 19.94 (41.37) for SNLI and 12.96 (38.56) for SST.

We also manually inspect the learned structures. We observe that in SNLI, the trees exhibit overall left-branching structure, which explains why the F_1 scores are closer to a left branching tree structure. Note that in our experiments on this corpus, the supervised syntax model does not perform as well as the left-to-right model, which suggests why the latent syntax model tends to converge towards the left-to-right model. We handpicked two examples of trees learned by our model and show them in Figure 2. We can see that in some cases the model is able to discover concepts such as noun phrases (e.g., *a boy*, *his sleds*) and simple verb phrases (e.g., *wearing sunglasses*, *is frowning*). Of course, the model sometimes settles on structures that make little sense to humans. We show two such examples in Figure 3, where the model chooses to compose *playing frisbee in* and *outside a* as phrases.

Training Time A major limitation of our proposed model is that it takes much longer to train compared to models with predefined structures. We observe that our models only outperforms models with fixed structures after several training epochs; and on some datasets such as SNLI or IMDB, an epoch could take a 5-7 hours (we use batch size 1 since the computation graph needs to be reconstructed for every example at every iteration depending on the samples from the policy network). This is also the main reason that we could only use smaller 100-dimensional Tree LSTM models in all our experiments. While for smaller datasets such as SICK the overall training time is approximately 6 hours, for SNLI or IMDB it takes 3-4 days for the model to reach convergence. In general, the latent syntax model and semi-supervised syntax models take about two or three times longer to converge compared to models with predefined structures.

5 Conclusion

We presented a reinforcement learning method to learn hierarchical structures of natural language sentences. We demonstrated the benefit of learning task-specific composition order on four tasks: sentiment analysis, semantic relatedness, natural language inference, and sentence generation. We qualitatively and quantitatively analyzed the induced trees and showed that they both incorporate some linguistically intuitive structures (e.g., noun phrases, simple verb phrases) and are different than conventional English syntactic structures.

REFERENCES

Johannes Bjerva, Johan Bos, Rob van der Goot, and Malvina Nissim. The meaning factory: Formal semantics for recognizing textual entailment and determining semantic similarity. In *Proc. of SemEval*, 2014.

Phil Blunsom and Trevor Cohn. Unsupervised induction of tree substitution grammars for dependency parsing. In *Proc. of EMNLP*, 2010.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proc. of EMNLP*, 2015.

Samuel R. Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D. Manning, and Christopher Potts. A fast unified model for parsing and sentence understanding. In *Proc. of ACL*, 2016.

David Chiang. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–228, 2007.

Kyunghyun Cho, Bart van Merriënboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint*, 2014.

Noam Chomsky. *Syntactic Structures*. Mouton, 1957.

Stephen Clark, Bob Coecke, and Mehrnoosh Sadrzadeh. A compositional distributional model of meaning. In *Proc. of the Second Symposium on Quantum Interaction*, 2008.

Qiming Diao, Minghui Qiu, Chao-Yuan Wu, Alexander J. Smola, Jing Jiang, and Chong Wang. Jointly modeling aspects, ratings and sentiments for movie recommendation (JMARS). In *Proc. of KDD*, 2014.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. Recurrent neural network grammars. In *Proc. of NAACL*, 2016.

Edward Grefenstette and Mehrnoosh Sadrzadeh. Experimental support for a categorical compositional distributional model of meaning. In *Proc. of EMNLP*, 2011.

Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997.

Sergio Jimenez, George Duenas, Julia Baquero, Alexander Gelbukh, Av Juan Dios Batiz, and Av Mendizabal. UNAL-NLP: Combining soft cardinality features for semantic textual similarity, relatedness and entailment. In *Proc. of SemEval*, 2014.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *Prof. of ACL*, 2014.

Yoon Kim. Convolutional neural networks for sentence classification. In *Proc. EMNLP*, 2014.

Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. Skip-thought vectors. In *Proc. of NIPS*, 2015.

Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *Proc. of ACL*, 2003.

Dan Klein and Christopher D. Manning. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proc. of ACL*, 2004.

Alice Lai and Julia Hockenmaier. Illinois-lh: A denotational and distributional approach to semantics. In *Proc. of SemEval*, 2014.

Mingbo Ma, Liang Huang, Bing Xiang, and Bowen Zhou. Dependency-based convolutional neural networks for sentence embedding. In *Proc. ACL*, 2015.

Marco Marelli, Luisa Bentivogli, Marco Baroni, Raffaella Bernardi, Stefano Menini, and Roberto Zamparelli. Evaluation of compositional distributional semantic models on full sentences through semantic relatedness and textual entailment. In *Proc. of SemEval*, 2014.

Lili Mou, Rui Men, Ge Li, Yan Xu, Lu Zhang, Rui Yan, and Zhi Jin. Natural language inference by tree-based convolution and heuristic matching. In *Proc. of ACL*, 2016.

Tsendsuren Munkhdalai and Hong Yu. Neural semantic encoders. *arXiv preprint*, 2016a.

Tsendsuren Munkhdalai and Hong Yu. Neural tree indexers for text understanding. *arXiv preprint*, 2016b.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Proc. of EMNLP*, 2014.

Steven Pinker. *Language Learnability and Language Development*. Harvard, 1984.

Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proc. of EMNLP*, 2011.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proc. of EMNLP*, 2013.

Richard Socher, Andrej Karpathy, Quoc V Le, Christopher D Manning, and Andrew Y Ng. Grounded compositional semantics for finding and describing images with sentences. *Transactions of the Association for Computational Linguistics*, 2:207–208, 2014.

Valentin I. Spitkovsky, Hiyan Alshawi, Angel X. Chang, and Daniel Jurafsky. Unsupervised dependency parsing without gold part-of-speech tags. In *Proc. of EMNLP*, 2011.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proc. NIPS*, 2014.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proc. of ACL*, 2015.

Ivan Vendrov, Ryan Kiros, Sanja Fidler, and Raquel Urtasun. Order-embeddings of images and language. In *Proc. of ICLR*, 2016.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

Luke S. Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proc. of UAI*, 2005.